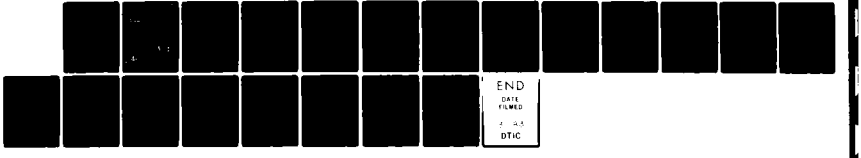
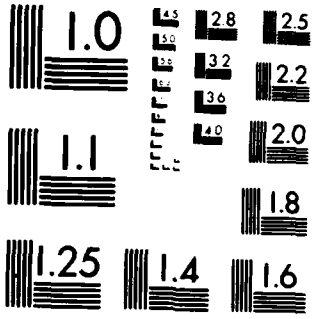
AD-A124 280 WHAT IS A SOFTWARE ENGINEERING ENVIRONMENT (SEE) 1/1
(DESIRED CHARACTERISTICS)(U) NAVAL SURFACE WEAPONS
CENTER DAHLGREN VA W P WARNER DEC 82 NSWC/TR-82-465
UNCLASSIFIED F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AD A124280

NSWC TR 82-465

WHAT IS A SOFTWARE ENGINEERING ENVIRONMENT (SEE)? (DESIRED CHARACTERISTICS)

BY WALTER P. WARNER

STRATEGIC SYSTEMS DEPARTMENT

DECEMBER 1982

Approved for public release; distribution unlimited.

DTIC ELECTE FEB 09 1983 S D E

DTIC FILE COPY

NAVAL SURFACE WEAPONS CENTER

Dahlgren, Virginia 22448 • Silver Spring, Maryland 20910

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: NSWC/TR-82-465 — 2. GOVT ACCESSION NO.: AD-A124280	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)*: WHAT IS A SOFTWARE ENGINEERING ENVIRONMENT (SEE)? (Desired Characteristics)	5. TYPE OF REPORT & PERIOD COVERED: Final
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Walter P. Warner	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Naval Surface Weapons Center (Code K04), Dahlgren, Virginia 22448	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: NIF
11. CONTROLLING OFFICE NAME AND ADDRESS: Naval Surface Weapons Center (Code K04), Dahlgren, Virginia 22448	12. REPORT DATE: December 1982
	13. NUMBER OF PAGES: 21
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*	15. SECURITY CLASS. *(of this report)*: UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Software
Software engineering
Environments
Tools
Software management

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

The technology of developing software has made significant advances in the past several years. Some supervisors, however, still treat it as if it were a "black art" and have no notion of how much progress is being made in the development of computer programs. This can only mean that the software effort is being performed by archaic methods and without the proper support tools. This report defines an "environment" for developing software. A proper "environment" not only aids the developer of the software but also

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

helps the supervisor in managing the development. The various phases of software development are described briefly and the characteristics of a facility to support the development in each phase are indicated.

FOREWORD

This work was conducted as a part of NSWC's continual effort to better understand and improve its methods of developing computer software. The preliminary draft of this report was reviewed by several of the Center's personnel, knowledgeable in the field, who made many valuable contributions.

Released by:

O. F. Braxton, Head
Strategic Systems Department

COPY INSPECTED

Accession For	
NTIS GRA&I	X
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A	

CONTENTS

PRECEDING PAGE BLANK-NOT FILMED

BACKGROUND

The technology of developing software has made significant advances in the past several years. Some supervisors, however, still treat it as if it were a "black art" and have no notion of how much progress is being made in the development of computer programs. This can only mean that the software effort is being performed by archaic methods and without the proper support tools. This report defines an "environment" for developing software. A proper "environment" not only aids the developer of the software but also helps the supervisor in managing the development and supports the creation of maintainable computer programs.

The cost of developing computer software has become one of the major expenditures of doing business today. According to the Government Accounting Office, two-thirds of all federal ADP spending is for software and related services<1>. The Department of Defense spent $3 billion for software in 1980<2>. A study in 1979 showed that NSWC consumed approximately 610 staff-years in the development of computer software delivered to the Navy and Marine Corps; this increased to 640 staff-years in 1980. In 1981, there were 1800 "user numbers" for the Center's mainframe computers. Thus one can see that in the Federal Government and at the Naval Surface Weapons Center the development of software is big business.

Studies of software projects have shown that 67% of the life-cycle cost of a software project (i.e., "cradle to the grave") is spent on maintenance <1>. Maintenance is defined as correcting errors and adding enhancements. It therefore seems that major consideration during development should be given those things that will make maintaining the software less costly and time consuming. This is the philosophy which will be used in defining the requirements for a generalized SOFTWARE ENGINEERING ENVIRONMENT (SEE).

Software engineering encompasses procedures, practices, and tools which support the development of computer software. These ideas did not appear to anyone in a flash of enlightenment as a completed package with the title "Software Engineering" printed on the front of it. "Good" programmers have used many of these ideas for years. The author can recall using the concept of "hierarchal decomposition" back in the late 1950's. These ideas have recently been put together in a package and the name "software engineering" has been attached to them.

The software engineering environment described in this report is an ideal system. All software development does not warrant the cost of developing a system of this magnitude. It is true, however, that we have always greatly underestimated the amount of effort required to develop software and the more costly the system the more important a

complete environment becomes. The more disciplined the effort becomes and the more support that is provided by automated tools the greater the cost savings over the life of the software. If an environment is already in place it is cost effective to use it for all but the smallest developments. The underlying concepts and principles of software engineering should be applied on all software efforts whether the work is to be done by one person or many.

The system described provides support for all phases of software development from conception of a system to solve a problem to support of the software while in operation. Since different projects may get to the Center in different phases of development, the environment as described may not apply in its entirety to all projects.

DEFINITION

When the term SOFTWARE ENGINEERING was first used, it was intended to draw attention to the fact that the development of software possessed neither the theoretical basis nor the discipline of engineering fields. The term was also intended to contrast with the term "computer science", which was perceived to be more concerned with defining the underlying principles of the application of computers and software. Software engineering is concerned with the actual development of software. The most widely accepted definition of software engineering is:

> The establishment and use of sound engineering principles in order to obtain, economically, software that is reliable and works efficiently on real machines. F. L. Bauer<3>

Webster defines environment as "the surrounding conditions or influences." A SOFTWARE ENGINEERING ENVIRONMENT (SEE) is therefore defined as the set of all the tools necessary to develop computer software using the principles of "software engineering." These tools should be automated (on a computer), integrated (the output of each tool in a format compatible with the input of each of its logical successors), and user friendly. As someone once put it, "a software engineering environment is a computer-aided design system for software."

SOFTWARE DEVELOPMENT AND MAINTENANCE PROBLEMS

Many of the problems with software have been attributed to poor management practices during the development phases. This is probably true; however, it has not always been the fault of the manager. In the past, software development has been a very individualized effort: the quality of the product depended totally on the ability of the person doing the development. The individual steps in the process were so interwoven that it was generally impossible to chart the actual progress. The specifications were usually determined, or at least extensively modified, while the design was being done. The design and the coding were often done at the same time; that is really to say that there was no design. The coding was a monolith and errors corrected in one section fouled up other sections of the code. No one therefore had any idea how close the project was to completion. The old adage of 90% complete for 90% of the project was thought to be a reality because the programmer himself really could not determine how close he was to being finished. The documentation usually was never started until all of the other work was finished and at that point even the programmer had forgotten how he/she had implemented many of the functions. It is impossible to manage such a task and the fact that some software projects were successful was only due to the fact that good people were doing them and the manager was lucky.

The underlying causes of poor management mentioned above are the same things that cause software to be very difficult and costly to maintain. First, coding is not an easy thing to understand at best. The fact that the documentation was poor meant that the maintenance people were on their own with little help. That is why it was so important to have the developers maintain the software. The fact that the coding constituted a monolith with interdependencies stretching from beginning to end made it very difficult to modify the code without introducing new errors.

SOFTWARE ENGINEERING PHILOSOPHY

The philosophy of software engineering is to apply good, well-defined, logical procedures (what is meant by engineering principles) to the development of software and to create documentation along with each step in the development. A part of those procedures creates the code as modules with no dependencies outside their own boundaries. This breaks large problems into a series of smaller ones that can be better understood and managed. All documentation is produced while the concepts are fresh. It is maintained on a computer for easy reference and updating and every item can be traced through all levels of documentation, right down to the code and test results.

One of the fundamental goals of software engineering is to identify and eliminate errors as early in the life cycle as possible. Figure 1 illustrates why this is important. The further into the development of the software, the more costly it is to remove an error. This is rather obvious since the further along in the development the more effort has been expended on each item (eg. design, implementation, test, documentation, etc.).

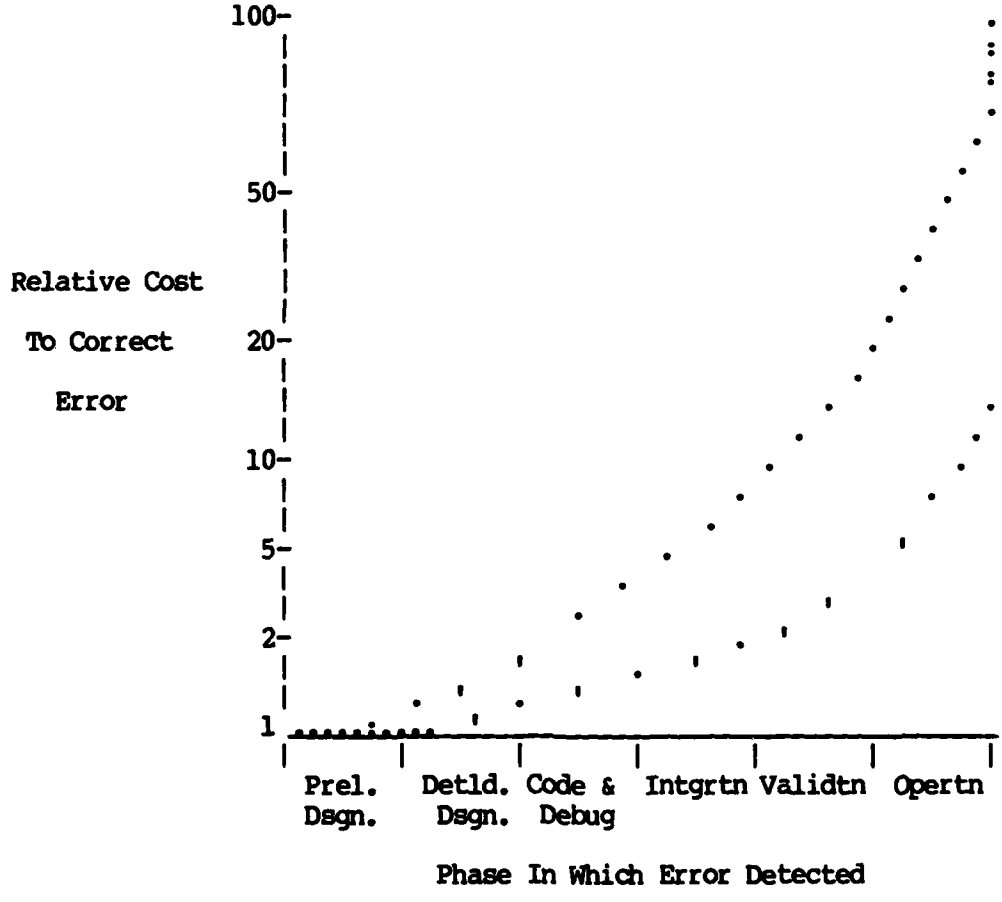


FIGURE 1. ERROR CORRECTION COSTS <4>

GENERAL REQUIREMENTS

A Software Engineering Environment should fully integrate development and communication tools. It should have the following features:

1. A single, compatible storage format for all documents, programs, and data files, so they can form a fully integrated data base

2. Software tools where the output of one is in the form of input for sucessive tools

3. Facilities for reporting and supervision, that allow quick surveys of progress on schedules and specifications without special efforts by supervisors or programmers to generate those surveys

4. All reports from the system designed for readability and understanding

5. Facilities for communication among programmers, including both electronic mail and "voice mail" (electronic storage and forwarding of voice messages)

6. Facilities which force disciplined software development proceeding directly from the specification and planning documents

7. Means of updating and annotating programs without destroying the original, so that a history of how the program has evolved is preserved

8. Provision for creating pointers that connect new documents or programs to existing information

9. An information-display management system with which a user can examine and cross-section the available information base. For example, the user will be able to view simultaneously a statement in assembly language, the high-level language statement that generated it, the associated portion of the design, and the original specification of its function.

10. Documentation for publication, derived straightforwardly and semi-automatically from the integrated data base. Documentation will not be a separate and belated activity.

THE SOFTWARE LIFE-CYCLE

The stages of a software life-cycle can be defined as follows:

Concept Development

Requirements Specification

Software System Design

Coding and Checkout

Testing

Integration

Operational Test and Evaluation

Deployment/Maintenance<5>.

CONCEPT DEVELOPMENT

The concepts phase consists of defining the problem that is to be solved. The primary need in this phase is to provide clear, concise documentation. The problems and subproblems should be specified in such a way that it will be possible to trace the solution through each level of subsequent documentation. This will consist of a computer-assisted documentation system which allows and requires unique identification of each problem. At this point consideration of whether the solution will be totally hardware, totally software run on a general-purpose computer, or a computer embedded in a system with appropriate software, is not germane. What often happens is that feasibility studies are not made and we build solutions for problems we don't understand and which often are the wrong problems.

The approach to doing analysis must be as disciplined as that for doing coding. There are several methods and techniques available which come under the general heading of structured analysis. Specific techniques may work best on a particular type of problem. No one technique will fit all of our work. Tools should be provided to support those techniques which are needed for our applications.

The output of this phase will be a good, well-analyzed, complete description of the problem to be solved. It will be stored in a computer and each aspect of the problem will be uniquely identified.

In many cases one's first association with a project will be after the concept has already been developed. If this is the case, one has the choice of putting the "concepts documentation" in

computer-readable form or ignoring that and starting to automate at the requirements specification phase.

REQUIREMENTS SPECIFICATION

R. Tausworthe of JPL has defined requirements in the following way:

> A requirement is a statement which clearly and accurately describes the essential technical features of a needed capability, along with a set of goals, constraints, criteria and conditions to be met.

What one must do is to determine those functions which must be carried out in order to achieve a solution to the problem. The output of this phase should tell what must be done to solve the problem but not how it is to be done. Which of the functions will be done by hardware, by a computer and its associated software, or by a human being will be specified. The resources necessary to complete the project and the high-risk areas will be identified.

It is important that an effective solution be found before any implementation is attempted. Therefore tools are needed which will not only assist in the determination of alternate solutions but will compare the efficacy of those solutions. The object will be to determine how each proposed solution will operate if it is implemented. This can be done by modeling, simulating at selected levels of detail, studying the timing relationships of the functions in the proposed solutions, by some means of rapidly prototyping proposed solutions, or some combination of these.

The SEE should retain each of the proposed solutions and its evaluation for future consideration of proposed modifications. The accepted solution should be identified for easy access. Again, as for all other levels of system description, each function should be uniquely labeled and be traceable to higher and subsequent lower levels of documentation. Each function will have identified a set of tests which will validate its implemention in the system. The flow of data and control between functions will be stored for graphical display (Functional Flow Diagram).

The following are functions that should be supported by the SEE during this phase:

1. A computer-processed language for describing the requirements, abstract data structures, and test requirements

2. Tools for modeling, simulation, and rapid prototyping, which will take the specification of the system to be modelled from the database

3. Tools which will assist in the determination of resources to carry out the project. This will consist of tools for determining the amount of physical resources, the number of staff-months, and the number of calendar months.

SOFTWARE SYSTEM DESIGN

In this phase the design, or structure, of the software is developed. The attempt will be to break the system down into modules which are small enough to be understood easily, contain only functions which are very closely related and dependent on each other, and do not depend on any other module except for data. The goal is to design a system so that any future changes will influence only a small number of modules and those modules will be easily identifiable.

There are two designs to consider in this phase. There is the design of the sequences of the functional modules and the design, or layout, of the data.

The following are the functions that should be supported by the SEE during this phase:

1. A computer-processed language for describing the design which provides for a "readable" listing of the design, consistency checking, completeness checking, and traceability of functions between higher- and lower-level documentation

2. Tools for graphically displaying the hierarchy and the flow of control of the modules

3. Tools for graphically displaying the flow of data between the modules, generally called a data flow diagram

4. Tools for constructing and modifying the structure of the data for the application programs, which will prevent duplicate identifiers and will show the relationship between each piece of data and the modules.

5. Since the software system will be developed incrementally the environment must have the capability to add to existing designs and to identify and store alternate designs (configuration management).

CODING AND CHECKOUT

During coding, the environment must not only support the new software development techniques but as much as possible force structure on the resultant code and check for conformity to coding standards. The system should encourage interactive generation of the code as opposed to writing it down to be keypunched. Automatic translation of the design language into code should be provided as extensively as possible. The system should assist in the correlation of coding segments to higher-level documentation. Prompts should be given for in-line documentation, such as prologues (explanatory material at the beginning of each module/routine). Facilities for on-line debugging should be provided. Statistics on the amount of code generated and the number of modules coded, debugged, and turned over to the configuration management team should be maintained by the system.

The SEE should provide the following capabilities:

1. The usual system routines needed to develop code, such as compilers, linkers, loaders, etc.

2. Tools for interactive development of software; full screen editors, syntax directed editors, etc.

3. Simulation of the target machine if not the same as the host

4. Source language debugging

5. Instrumentation of the code to provide data extraction when code is executed

6. Context cross reference, i.e., to be able to review higher levels of documentation which relate to the section of code being implemented. The system should determine through its traceability features which documentation is relevant and display it on command.

7. Metrics which indicate the size, complexity<6><7>, performanace, etc. of the modules

8. Analysis tools such as set/use maps, cross reference maps, etc.

9. Facilities for generating stubs for testing the flow of control prior to the existence of all modules

10. Static analyzers which check for conformance to standards, which should be both general and project specific

11. Error seeding; the ability to insert errors to test whether the program will operate properly under error conditions

TESTING

If one has a separate test group, and this is recommended, the computer programs making up the system will be turned over to this group by the developer and the programs must be put together and tested as a system. If a separate test group does not exist, then it is advisable to have someone who acts as a Program Secretary who is responsible for all official versions of the software. It should now come under official control so that changes will not be made to it without proper controls. The programs making up the system probably will be tested one at a time and the tested programs then incorporated into the system. Stubs may be generated for those units not complete and tests run on the resultant system. Since there may be errors in the units requiring them to be turned back to the programmer for correction, there must be a capability of keeping track of each version of the units. There must also be a means of identifying which versions of the units are in each subset of the software system tested. Some of the capabilities identified under Coding and Checkout may be used during this phase in order to provide final certification. The development group may also want to use some of the capabilities identified for this phase in order to forestall future problems.

The SEE must have the following capabilities to support testing:

1. Static code analyzers to determine conformance to standards, etc.

2. Test scenario generation, test case generation if technologically feasible.

3. The ability to reference previous test data, make changes to it, and identify this as another test case.

4. Automatic comparison of the results of runs and reporting of the differences.

5. Facilities for "instrumenting" the program for data and event extraction during execution. When commanded, the system should automatically instrument the code for use in test coverage analysis.

6. Test reporting which includes test coverage, tested and untested requirements, etc.

CONFIGURATION MANAGEMENT

The problems of configuration management are keeping track of modifications to the baselined software, making sure that none are made without formal approval, knowing precisely the capabilities of each version of the software system, and knowing which sites (if more than one) have which version at any given time.

To carry out this function, the environment must have a data base management system (DBMS) with the capability of generating good quality reports. As mentioned in the section on general requirements, the environment should have a fully integrated data base which contains all of the information generated from the beginning of concept development to the death of the program. The PROJECT DATA BASE will be the foundation on which everything in the environment will be built. There will need to be a command language which can be used to create a system of the software from the version numbers of the modules. There must be convenient methods for keeping track of change orders, trouble/failure reports, and their status.

PROJECT MANAGEMENT CONSIDERATIONS

The project managers must be able to query all data in the data base needed to manage the project. This includes data needed to determine project schedules and status, product quality, etc. The data should be presented in the form of quality reports rather than isolated data.

The environment should also have the following capabilities:

1. Management planning, scheduling and tracking tools; for example, an automated Work Breakdown Structure tool

2. Generating, storing, and retrieving management information such as progress reports

3. The ability to generate quality summary reports from data in any area of the data base.

REFERENCES

<1> "Special Report on Software and Services", DATAMATION, Aug 25, 1981, p66.

<2> DOD Annual Report FY 81, 29 Jan 1981, p245.

<3> F. L. Bauer, "Software Engineering," Information Processing 71 (1972), North Holland Publishing Co., pp530.

<4> Jensen, R. W., C. C. Tonies, Software Engineering, Prentice Hall, Inc., 1979, p212.

<5> "Computer Software Life Cycle Management Guide," NAVELEXINST 5200.23, 1 Mar 1979, p xviii.

<6> Harrison, et. al.,"Applying Software Complexity Metrics to Program Maintenance," COMPUTER, Sept 1982.

<7> J. C. Zolnowski, D. B. Simmons; "Taking the Measure of Program Complexity," AFIPS Conference Proceedings, Vol. 50, 1981 National Computer Conference, pp. 329-336, 1981.

BIBLIOGRAPHY

Greenstein, J. S., R. C. and B. H. Willeges, "Human-Computer Dialogue Design: Hardware and Software," 1981 Fall Industrial Engineering Conference Proceedings, Industrial Engineering & Management Press, Norcross, Ga. 30092.

Hausen, H., Mullerburg, M., Riddle, W. E., Software Engineering Environments: A Bibliography, S. Hunke, Editor, North Holland Publishing Co., 1981.

Munson, J. B, "Software Maintainability: A Practical Concern for Life Cycle Costs," IEEE Computer, Nov 1981.

Singer, A., H. Ledgard, and J. F. Hueras, "The Annotated Assistant; A Step Towards Human Engineering," IEEE Transactions on Software Engineering, Vol. SE-7, No. 4, July 1981.

Stuebing, H. G., A Modern Facility for Software Production and Maintenance, NADC, Warminister, Pa. 18974.

Willeges, B. H. and R. C. Willeges, "User Considerations in Computer-Based Information Systems," VPI&SU, Report CSIE-81-2, Sept 81.

DISTRIBUTION

Computer Science Department
VPI&SU Blacksburg, Va. 24061
(Drs. Nance, Hartson, Lindquist)

Computer Science Department
U. of Maryland
College Park , Maryland 20742
(Dr. V. Basili)

Dept of Math and Computer Science
James Madison University
Harrisonburg, Va. 22807

Dept of Applied Math and Computer Sciences
University of Virginia
Charlottesville, Va. 22901
(Dr. J. Ortega)

University of Seattle
Dept of Software Engineering
Seattle, Wash. 98122
(Dr. Kyu Y. Lee)

Dept of Math and Computer Sciences
Mary Washington College
Fredericksburg, Va. 22401

Defense Technical Information Center (12)

Local: List C-1

Dahlgren;
- F54 P. Brown
- G12 L. Batayte
- K04 20 copies
- K105
- K14
- K301
- K33 Flink
- K51 (2 copies)
- K52 J. Smith, J. Dooley
- K53 Huber
- K54 McCoy
- N20A, N20E
- N21 M. Masters, D. McConnell
- N22 R. Crowder
- N31 G. Stout
- N34 J. Lynch
- N51 Harrison
- E431 (6 copies)

White Oak;
- E346
- K34 (2 copies)
- M20
- R 42 G. Powell
- U22 H. Cook
- U23 J. Cottrell
- E432 (2 copies)

END

DATE
FILMED

3-83

DTIC